HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1

BÁO CÁO BÀI TẬP LỚN

MÔN HỌC: KIẾN TRÚC VÀ THIẾT KẾ PHẦN MỀM

Giảng viên: Trần Đình Quế

Nhóm lớp học: INT1342M nhóm 06

Nhóm bài tập lớn: 12

Sinh viên: Đinh Minh Phúc

Mã sinh viên: B20DCCN503

Hà Nội, 2024

I. Mô tả hệ thống bằng ngôn ngữ tự nhiên

1. Module quản lý sách

- Mục đích của modul này là quản lý thông tin của sách.
- Tính năng của Module:
  + Lấy danh sách nhà xuất bản. Thêm, sửa, xóa thông tin nhà xuất bản
  + Lấy danh sách tác giả. Thêm, sửa, xóa thông tin tác giả
  + Lấy danh sách của sách. Thêm, sửa, xóa thông tin sách

2. Module quản lý mobile

- Mục đích chính của module này là quản lý thông tin của mobile
- Tính năng của mobile:
  + Lấy danh sách nhà sản xuất. Thêm, sửa, xóa thông tin nhà sản xuất
  + Lấy danh sách kiểu sản phẩm. Thêm, sửa, xóa thông tin kiểu sản phẩm
  + Lấy danh sách của mobule. Thêm, sửa, xóa thông tin của mobile

3. Module quản lý clothe

- Mục đích chính của module này là quản lý thông tin của clothe
- Tính năng của clothe:
  + Lấy danh sách nhà sản xuất. Thêm, sửa, xóa thông tin nhà sản xuất
  + Lấy danh sách phong cách thời trang. Thêm, sửa, xóa thông tin phong cách thời trang
  + Lấy danh sách của clothe. Thêm, sửa, xóa thông tin của clothe

4. Module catalog

- Mục đích chính của module này cho phép người dùng tìm kiếm, lựa chọn sản phẩm mình muốn cho vào giỏ hang
- Tính năng của module:
  + Tìm kiếm sản phẩm theo tên
  + Chọn sản phẩm vào giỏ hang

5. Module Cart

- Mục đích chính của module này cho phép người dùng quản lý giỏ hàng của mình
- Tính năng của module:
  + Chỉnh sửa số lượng sản phẩm
  + Xóa sản phẩm khỏi giỏ hang
  + Chọn các sản phẩm để tạo order

6. Module inventory

- Mục đích chính của module này là để kiểm tra xem số lượng sản phẩm mà người dùng muốn thêm vào giỏ hàng hoặc đặt hàng có hợp lệ hay không
- Chức năng chính của module:
  + Kiểm tra số lượng sản phẩm có lớn hơn số lượng còn lại trong kho không
  + Kiểm tra xem sản phẩm này có tồn tại hoặc có thể đặt hàng/thêm giỏ hàng không

7. Module Order

- Mục đích chính của module này để quản lý danh sách các order của người dùng
- Chức năng của module:
  + Lấy danh sách các order theo tình trạng(mới được tạo, đã thanh toán, chờ vận chuyển, đã hoàn thành)
  + Xác nhận thanh toán cho đơn hàng
  + Hủy 1 đơn hàng

8. Module Payment

- Module này cho phép người dùng lựa chọn thanh toán cho 1 order
- Chức năng chính của module:
  + Thanh toán cho 1 order, đưa ra thông tin hóa đơn

9. Module search

- Module này cho phép người dùng tìm kiếm thông tin sản phẩm
- Chức năng của module:

+ Tìm kiếm book theo từ khóa
+ Tìm kiếm mobile theo từ khóa
+ Tìm kiếm clothe theo từ khóa

10. Module user

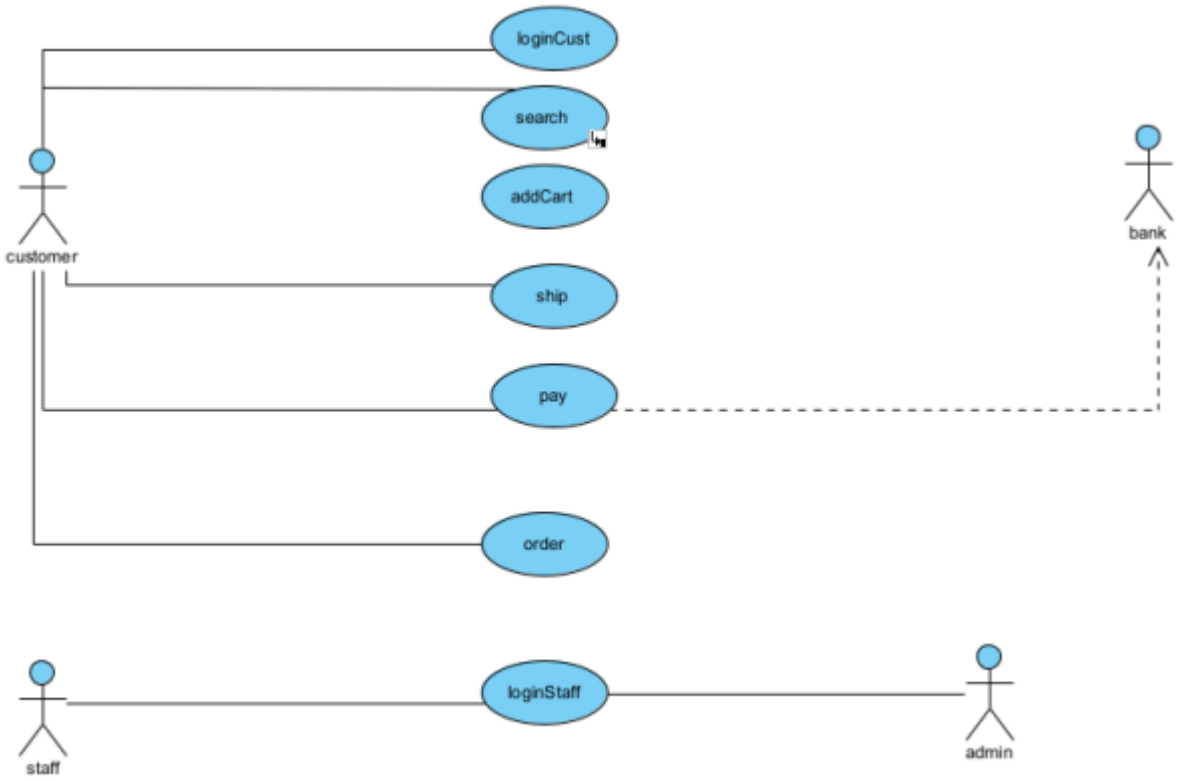
- Module này cho phép người dùng quản lý thông tin tài khoản của mình
- Chức năng của module:
  + Đăng ký tài khoản
  + Đăng nhập hệ thống
  + Đổi mật khẩu
  + Quên mật khẩu

I. Biểu đồ usecase tổng quát và biểu đồ usecase chi tiết cho từng dịch vụ/chức năng

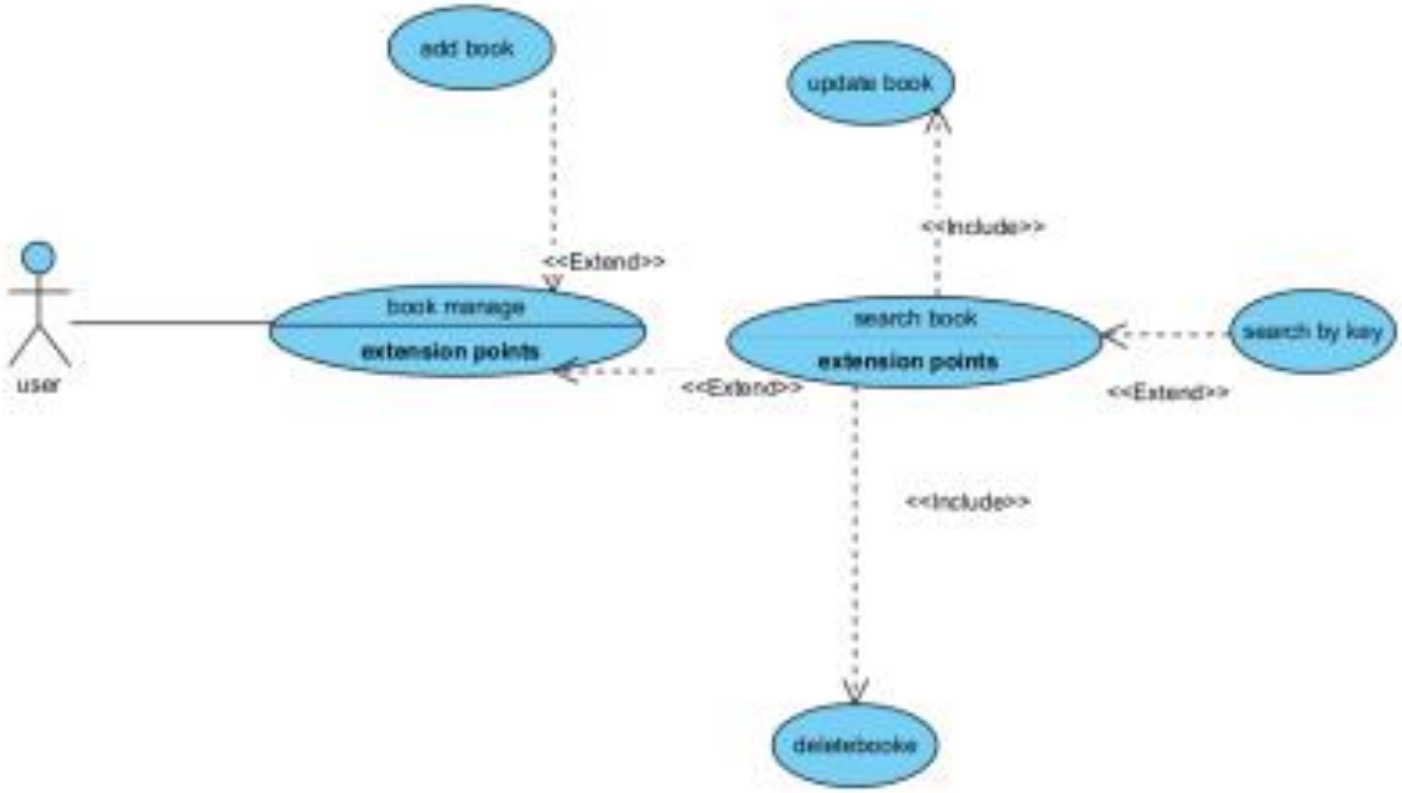
Actor	Functionalities	Detailed Description
Customer	Search	- Khách nhập từ khóa vào thanh tìm kiếm và nhấn Enter. - Hệ thống hiển thị danh sách các sản phẩm phù hợp với từ khóa tìm kiếm. - Khách chọn một sản phẩm từ danh sách hiển thị. - Hệ thống chuyển hướng Khách đến trang chi tiết sản phẩm để xem thông tin chi tiết.
	Add to cart	- Tại trang chi tiết sản phẩm, Khách nhấn nút "Thêm vào giỏ hàng". - Hệ thống thêm sản phẩm vào giỏ hàng của Khách và hiển thị thông báo xác nhận.
	Pay	- Sau khi Khách đã thêm sản phẩm vào giỏ hàng, Khách nhấn vào biểu tượng giỏ hàng để tiến hành thanh toán. - Hệ thống hiển thị trang giỏ hàng với danh sách các sản phẩm đã chọn. - Khách kiểm tra lại thông tin giỏ hàng và chọn nút "Thanh toán". - Hệ thống chuyển hướng Khách đến trang thanh toán để nhập thông tin thanh toán.
	Order	- Tại trang thanh toán, Khách nhập thông tin thanh toán và giao hàng. - Khách chọn phương thức thanh toán (ví điện tử, thẻ tín dụng, chuyển khoản ngân hàng, v.v.). - Khách nhấn nút "Đặt hàng" để hoàn tất quy trình thanh toán. - Hệ thống hiển thị thông báo xác nhận và gửi email cho Khách xác nhận đơn hàng cùng với thông tin chi tiết.
Admin	Cập nhật thông tin cá nhân	- Khách click “Quản lý thông tin cá nhân”. - Hệ thống hiển thị giao diện quản lý thông tin của người dùng. - Nếu Khách click “Sửa thông tin”, hệ thống hiện giao diện sửa thông tin với các thông tin có thể sửa. - Khách sửa thông tin muốn sửa và click lưu. - Hệ thống cập nhật thông tin của Khách.
	Đăng nhập	- Khách nhập thông tin đăng nhập trên giao diện đăng nhập. - Hệ thống xác minh và thông báo thông tin đăng nhập cho người dùng.
	Đăng ký	- Khách click “Đăng ký”. – Hệ thống hiển thị giao diện đăng ký với các trường thông tin cần thiết. - Khách nhập các thông tin cần thiết. - Hệ thống xác minh thông tin và thông báo thông tin đăng ký cho Khách.
Shipment	Thêm thông tin vận chuyển	- Khách nhập địa chỉ mới và click lưu. - Hệ thống thêm địa chỉ vận chuyển mới của Khách.

		- Nếu Khách click sửa thông tin địa chỉ vận chuyển, hệ thống hiển thị giao diện sửa địa chỉ vận chuyển. - Khách sửa các thông tin về địa chỉ và click lưu. - Hệ thống cập nhật thông tin địa chỉ vận chuyển của Khách. - Nếu Khách click xóa thông tin địa chỉ vận chuyển, hệ thống xác minh và thông báo thông tin xóa địa chỉ vận chuyển cho Khách

General Use case Diagram

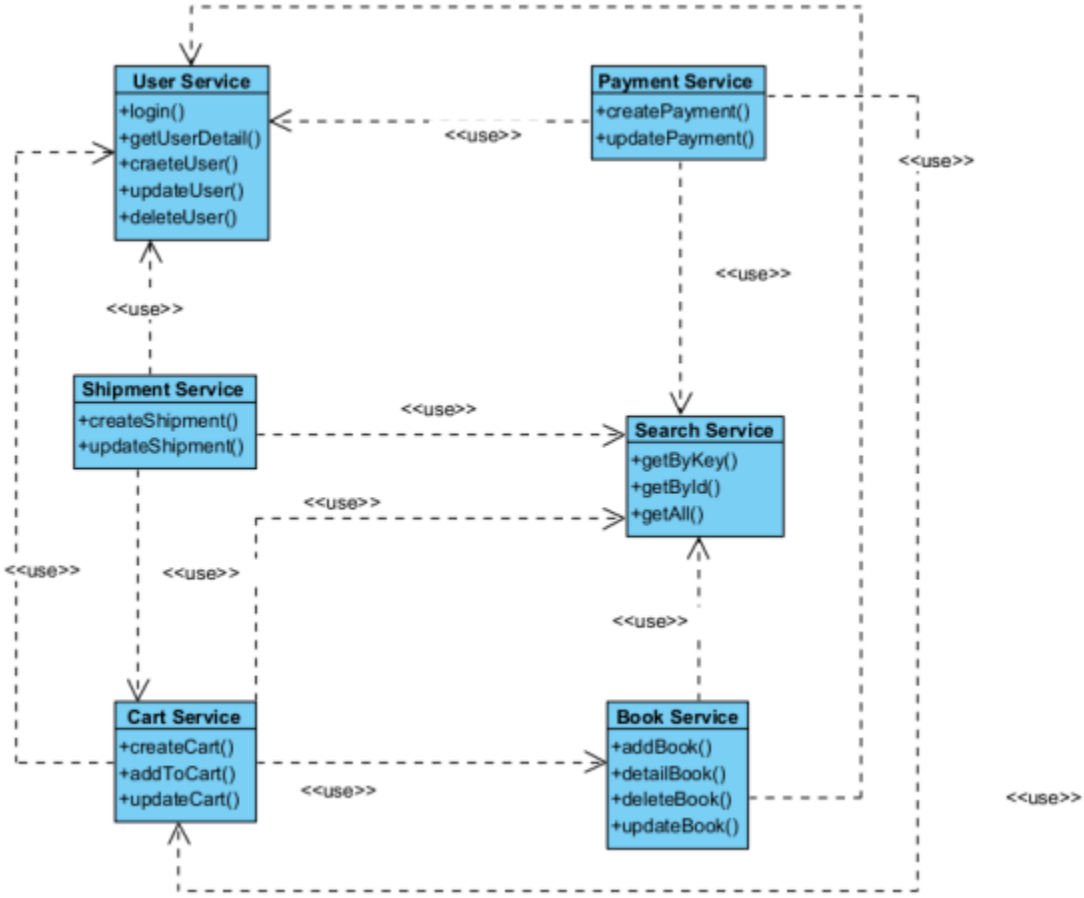


Detailed Use case for book



Detail for user

Decompose a software systems into module/services



II. Trình bày RestAPI và áp dụng cho hệ thống ecomSys

1. Các bước tạo API trong Cart

a) Tạo các class model

```python
class CartItem(models.Model):
    date_added = models.DateTimeField(auto_now_add = True)
    quantity = models.IntegerField(default = 1)
    product_id = models.IntegerField(default = 1)
    product_type = models.CharField(max_length = 30)
    user_id = models.IntegerField()

    class Meta:
        db_table = 'cart_item'
        ordering = ['date_added']
```

b) Tạo các class Serializer

```python
class CartJson(serializers.ModelSerializer):
    class Meta:
        model = CartItem
        fields = ['id', 'quantity', 'product_id', 'product_type', 'user_id']

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        request_method = self.context['request'].method
        if request_method == 'DELETE':
            self.fields['id'].required = True
            self.fields['quantity'].required = False
            self.fields['product_id'].required = False
            self.fields['product_type'].required = False

        self.fields['user_id'].required = False

    def validate_quantity(self, value):
        if value <= 0:
            raise serializers.ValidationError("Quantity must be greater than 0")
        return value

    def validate_product_type(self, value):
        if value != 'MOBILE' and value != 'CLOTHE' and value != 'BOOK':
            raise serializers.ValidationError("Product type not valid")
        return value
```

c) Tạo các class Form:

```python
class CartItemForm(forms.ModelForm):
    class Meta:
        model = CartItem
        fields = ['quantity', 'product_id']
```

d) Tạo class class api với các phương thức:

- Phương thức GET:

```python
@method_decorator(is_authenticated)
def get(self, request, format=None):
    user_id = request.session['user_id']
    cart_items = CartItem.objects.filter(user_id=user_id)
    result = []
    for cart in cart_items:
        cartResponse = {
            'id': cart.id,
            'quantity': cart.quantity,
            'product_id': cart.product_id,
            'product_type': cart.product_type,
        }

        try:
            if checkProductInStock(cartResponse, cartResponse):
                result.append(cartResponse)
        except ProductNotValidException:
            cart.delete()

    return Response(result, status=status.HTTP_200_OK)
```

- Phương thức POST:

```python
@method_decorator(is_authenticated)
def post(self, request, format=None):
    user_id = request.session['user_id']
    cart = CartJson(data = request.data, context={'request': request})
    if cart.is_valid():
        validated_data = cart.validated_data
        validated_data['user_id'] = user_id
        try:
            cart_stock = CartItem.objects.get(user_id = user_id, product_id = validated_data['product_id'], product_type = validated_data['pr
            cart_stock.quantity += validated_data['quantity']
            validated_data['quantity'] = cart_stock.quantity
            isFirst = False
        except ObjectDoesNotExist:
            cart_stock = cart
            isFirst = True

        try:
            if checkProductInStock(validated_data):
            #Nếu đã thêm sản phẩm này vào giỏ hàng thì sẽ công thêm số lượng, chưa có thì tạo mới
                cart_stock.save()
                return Response(status=status.HTTP_200_OK)
            else:
                return Response('Your number product in cart will more than number instock, please try again', status=status.HTTP_400_BAD_REQ

        except ProductNotValidException:
            if not isFirst: #Nếu như đã thêm sản phẩm vào giỏ hàng thì mới xóa
                cart_stock.delete()
            return Response('Product is not valid', status.HTTP_400_BAD_REQUEST)
    return Response(cart.errors, status=status.HTTP_400_BAD_REQUEST)
```

- Phương thức PUT:

```python
@method_decorator(is_authenticated)
def put(self, request, format=None):
    user_id = request.session['user_id']
    cart = CartJson(data = request.data, context={'request': request})
    if cart.is_valid():
        validated_data = cart.validated_data
        try:
            cart_stock = CartItem.objects.get(user_id = user_id, product_id = validated_data['product_id'], product_type = validated_data['pr
            cart_stock.quantity = validated_data['quantity']
            validated_data['quantity'] = cart_stock.quantity
            if cart_stock.quantity == 0:
                cart_stock.delete()
        except ObjectDoesNotExist:
            return Response("You haven't added this product to cart", status=status.HTTP_400_BAD_REQUEST)

        try:
            if checkProductInStock(validated_data):
                #Nếu số lượng mới = 0 thì sẽ xóa khỏi giỏ hàng
                cart_stock.save()
                return Response(status=status.HTTP_200_OK)
            else:
                return Response('Your number product in cart will more than number instock, please try again', status=status.HTTP_400_BAD_REQ
        except ProductNotValidException:
            cart_stock.delete()
            return Response('Product is not valid', status.HTTP_400_BAD_REQUEST)

    return Response(cart.errors, status=status.HTTP_400_BAD_REQUEST)
```

- Phương thức DELETE:

```python
@method_decorator(is_authenticated)
def delete(self, request, format = None):
    try:
        user_id = request.session['user_id']
        cartItem = CartItem.objects.get(id = request.data['id'], user_id = user_id)
        cartItem.delete()
    except ObjectDoesNotExist:
        return Response("This is not your cart", status=status.HTTP_400_BAD_REQUEST)
    return Response(status=status.HTTP_200_OK)
```

2. Các bước tạo API trong Book, Mobile

a) Book

- B1: Sử dụng câu lệnh để tạo app có tên book

```
PS C:\Users\PHUC\Desktop\django\ecomSys_cnpm6.13_phuc> python manage.py startapp book
```

- B2: Thêm app đó vào danh sách các app:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'cart',
    'book',
    'catalog',
    'djongo',
    'rest_framework',
    'search',
    'user',
    'order',
    'clothe',
    'management',
    'mobile',
    'payment',
    'shipment',
]
```

- B3: Cấu hình database cho app (Sử dụng MongoDB)

```
'book': {
    'ENGINE': 'djongo',
    'NAME': 'book',
    'CLIENT': {
        'host': 'localhost',
        'port': 27017,
    }
},
```

- B4: Cấu hình database_router cho app

```python
class BookRouter:
    def db_for_read(self, model, **hints):
        if model._meta.app_label == 'book':
            return 'book'
        return None

    def db_for_write(self, model, **hints):
        if model._meta.app_label == 'book':
            return 'book'
        return None

    def allow_relation(self, obj1, obj2, **hints):
        if obj1._meta.app_label == 'book' and obj2._meta.app_label == 'book':
            return 'book'
        return None

    def allow_migrate(self, db, app_label, model_name=None, **hints):
        if app_label == 'book':
            return 'book'
        return None
```

- B5: Cập nhật router đó vào danh sách router

```python
DATABASE_ROUTERS = [
    'book_store.database_router.BookRouter',
    'book_store.database_router.CartRouter',
    'book_store.database_router.UserRouter',
    'book_store.database_router.OrderRouter',
    'book_store.database_router.MobileRouter'
]
```

- B6: Cài đặt các url sẽ gọi tới app này trong urls.py

```python
urlpatterns = [
    path('admin/', admin.site.urls),

    #Nếu dùng như sau thì sẽ không cần book: path('', include('book.urls'))
    path('', include('catalog.urls')),
    path('cart', include('cart.urls')),
    path('search', include('search.urls')),
    path('user', include('user.urls')),
    path('order', include('order.urls')),
    path('management-book', include('book.urls')),
    path('management-clothe', include('clothe.urls')),
    path('management-mobile', include('mobile.urls')),
]
```

- B7: Tạo các class Model

```python
class Category(models.Model):
    name = models.CharField(max_length = 50)
    description = models.TextField()
    created_at = models.DateTimeField(auto_now_add = True)
    updated_at = models.DateTimeField(auto_now = True)

    class Meta:
        db_table = 'category'
        ordering = ['-created_at']

    def __str__(self):
        return self.name
```

```python
class Author(models.Model):
    name = models.CharField(max_length=100)
    description = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    class Meta:
        db_table = 'author'
        ordering = ['-created_at']

    def __str__(self):
        return self.name
```

```python
class Publisher(models.Model):
    name = models.CharField(max_length=100)
    description = models.TextField()
    address = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    class Meta:
        db_table = 'publisher'
        ordering = ['-created_at']

    def __str__(self):
        return self.name

class Book(models.Model):
    name = models.CharField(max_length=224, null=False, blank=False)
    price = models.IntegerField(default = 0)
    image = models.ImageField(upload_to=book_image_path)
    description = models.TextField(default='')
    created_at = models.DateTimeField( default=timezone.now)
    updated_at = models.DateTimeField(auto_now = True)
    categries = models.ManyToManyField(Category)
    is_active = models.BooleanField(default = True)
    authors = models.ManyToManyField(Author)
    publishers = models.ManyToManyField(Publisher)

    class Meta:
        db_table = 'book'
        ordering = ['-created_at']
```

- B7: Tạo các class Serializer:

```python
class AuthorJson(serializers.ModelSerializer):
    class Meta:
        model = Author
        fields = ['id', 'name']

class PublisherJson(serializers.ModelSerializer):
    class Meta:
        model = Publisher
        fields = ['id', 'name', 'address']

class CategoryJson(serializers.ModelSerializer):
    class Meta:
        model = Category
        fields = ['id', 'name']

class BookJson(serializers.ModelSerializer):
    categries = CategoryJson(many=True, read_only=True)
    authors = AuthorJson(many=True, read_only=True)
    publishers = PublisherJson(many=True, read_only=True)
    class Meta:
        model = Book
        fields = ['id', 'name', 'price', 'image', 'description', 'created_at', 'updated_at', 'categries', 'is_active', 'authors', 'publishers']
```

- B8: Viết class kế thừa từ APIView để tạo thực hiện các phương thức GET, POST,PUT,DELETE

```python
class BookApi(APIView):
```

- B9: Tạo phương thức GET

```python
@method_decorator(is_authenticated)
@method_decorator(is_admin)
def get(self, request, format=None):
    # Xử lý tìm kiếm
    query = request.GET.get('q')
    books = Book.objects.all()
    if query == 'null':
        query = ''
    print(query)
    books = books.filter(Q(name__icontains=query) | Q(description__icontains=query))

    # Phân trang
    paginator = Paginator(books, 10)  # Chia danh sách thành các trang, mỗi trang có tối đa 10 sách
    page = request.GET.get('page', 1)  # Trang mặc định là trang đầu tiên nếu không có trang được chỉ định
    try:
        books_page = paginator.page(page)
    except PageNotAnInteger:
        books_page = paginator.page(1)
    except EmptyPage:
        books_page = paginator.page(paginator.num_pages)

    # Tính toán totalPage và currentPage
    total_pages = paginator.num_pages
    current_page = books_page.number

    # Tạo dữ liệu cho serializer BookListSerializer
    serialized_data = {
        'books': books_page,
        'totalPage': total_pages,
        'currentPage': current_page,
    }

    # Serialize dữ liệu của serializer
    result_serializer = ListBookJson(serialized_data)
    return Response(result_serializer.data)
```

- B10: Tạo phương thức POST

```python
@method_decorator(is_authenticated)
@method_decorator(is_admin)
def post(self, request, format=None):
    book = BookJson(data = request.data)
    if book.is_valid():
        book.save()
        return Response(status=status.HTTP_200_OK)
    return Response(book.errors, status=status.HTTP_400_BAD_REQUEST)
```

- B11: Tạo phương thức PUT

```python
@method_decorator(is_authenticated)
@method_decorator(is_admin)
def put(self, request, format=None):
    id = request.data.get('id', None)
    book = get_object_or_404(Book, pk=id)
    old_image_path = book.image.path if book.image else None  # Lưu đường dẫn ảnh cũ

    form = BookJson(instance=book, data=request.data)
    if form.is_valid():
        # Kiểm tra xem có ảnh mới được gửi không
        if 'image' in request.FILES:
            # Xóa ảnh cũ đi
            if old_image_path and os.path.exists(old_image_path):
                os.remove(old_image_path)

        form.save()
        return Response(status=status.HTTP_200_OK)
    return Response(form.errors, status=status.HTTP_400_BAD_REQUEST)
```

- B12: Tạo phương thức DELETE

```python
@method_decorator(is_authenticated)
@method_decorator(is_admin)
def delete(self, request, format=None):
    id = request.data.get('id', None)
    book = get_object_or_404(Book, pk = id)
    old_image_path = book.image.path if book.image else None  # Lưu đường dẫn ảnh cũ
    book.delete()
    #Xóa ảnh đại diện
    if old_image_path and os.path.exists(old_image_path):
        os.remove(old_image_path)
    return Response(status=status.HTTP_200_OK)
```

- B13: Chỉ định url cho API này

```
urlpatterns = [
    path('/api/book', BookApi.as_view() , name='book-api'), #name là biến được gọi đến khi sử dụng url
    path('/', views.show_home_book, name='show-book-list'),
    path('/add', views.show_add_book, name='show-add-book'),
    path('/edit/<int:id>', views.show_edit_book, name='show-edit-book'),
    path('/delete/<int:id>', views.delete_book, name='delete-book'),

    path('/categories-list', views.show_categories, name='show-book-categories' ),
    path('/category-add', views.save_category, name='save-book-category'),
    path('/category-update/<int:id>', views.update_category, name='update-book-category'),
    path('/category-delete/<int:id>', views.delete_category, name='delete-book-category'),

    path('/author-list', views.show_authors, name='show-authors'),
    path('/author-add', views.save_author, name='save-author'),
    path('/author-update/<int:id>', views.update_author, name='update-author'),
    path('/author-delete/<int:id>', views.delete_author, name='delete-author'),

    path('/publishers-list', views.show_publishers, name='show-publishers'),
```

b) Mobile

- B1: Sử dụng câu lệnh để tạo app có tên mobile

```
PS C:\Users\PHUC\Desktop\django\ecomSys_cnpm6.13_phuc> python manage.py migrate --database=mobile
```

- B2: Thêm app đó vào danh sách các app:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'cart',
    'book',
    'catalog',
    'djongo',
    'rest_framework',
    'search',
    'user',
    'order',
    'clothe',
    'management',
    'mobile',
    'payment',
    'shipment',
]
```

- B3: Cấu hình database cho app (Sử dụng MongoDB)

```
'mobile': {
    'ENGINE': 'djongo',
    'NAME': 'mobile',
    'CLIENT': {
        'host': 'localhost',
        'port': 27017,
    }
},
```

- B4: Cấu hình database_router cho app

```python
class MobileRouter:
    def db_for_read(self, model, **hints):
        if model._meta.app_label == 'mobile':
            return 'mobile'
        return None

    def db_for_write(self, model, **hints):
        if model._meta.app_label == 'mobile':
            return 'mobile'
        return None

    def allow_relation(self, obj1, obj2, **hints):
        if obj1._meta.app_label == 'mobile' and obj2._meta.app_label == 'mobile':
            return 'mobile'
        return None

    def allow_migrate(self, db, app_label, model_name=None, **hints):
        if app_label == 'mobile':
            return 'mobile'
        return None
```

- B5: Cập nhật router đó vào danh sách router

```python
DATABASE_ROUTERS = [
    'book_store.database_router.BookRouter',
    'book_store.database_router.CartRouter',
    'book_store.database_router.UserRouter',
    'book_store.database_router.OrderRouter',
    'book_store.database_router.MobileRouter'
]
```

- B6: Cài đặt các url sẽ gọi tới app này trong urls.py

```
urlpatterns = [
    path('admin/', admin.site.urls),

    #Nếu dùng như sau thì sẽ không cần book: path('', include('book.urls'))
    path('', include('catalog.urls')),
    path('cart', include('cart.urls')),
    path('search', include('search.urls')),
    path('user', include('user.urls')),
    path('order', include('order.urls')),
    path('management-book', include('book.urls')),
    path('management-clothe', include('clothe.urls')),
    path('management-mobile', include('mobile.urls')),
]
```

- B7: Tạo các class Model

```
class Producer(models.Model):
    name = models.CharField(max_length=224, null=False, blank=False)
    description = models.TextField()
    created_at = models.DateTimeField( default=timezone.now)
    updated_at = models.DateTimeField(auto_now = True)
    class Meta:
        db_table = 'mobile_producer'
        ordering = ['-created_at']

    def __str__(self):
        return self.name

class Type(models.Model):
    name = models.CharField(max_length=224, null=False, blank=False)
    description = models.TextField()
    created_at = models.DateTimeField( default=timezone.now)
    updated_at = models.DateTimeField(auto_now = True)
    class Meta:
        db_table = 'type'
        ordering = ['-created_at']

    def __str__(self):
        return self.name
```

```python
class Mobile(models.Model):
    name = models.CharField(max_length=224, null=False, blank=False)
    price = models.IntegerField(default = 0)
    image = models.ImageField(upload_to=book_image_path)
    description = models.TextField(default='')
    created_at = models.DateTimeField( default=timezone.now)
    updated_at = models.DateTimeField(auto_now = True)
    is_active = models.BooleanField(default = True)
    types = models.ManyToManyField(Type)
    producers = models.ManyToManyField(Producer)

    class Meta:
        db_table = 'mobile'
        ordering = ['-created_at']
```

- B7: Tạo các class Serializer:

```python
class TypeJson(serializers.ModelSerializer):
    class Meta:
        model = Type
        fields = ['id', 'name', 'description']

class ProducerJson(serializers.ModelSerializer):
    class Meta:
        model = Producer
        fields = ['id', 'name', 'description']

class MobileJson(serializers.ModelSerializer):
    types = TypeJson(many=True, read_only=True)
    producers = ProducerJson(many=True, read_only=True)
    class Meta:
        model = Mobile
        fields = ['id', 'name', 'price', 'image', 'description', 'is_active', 'types', 'producers']
```

- B8: Viết class kế thừa từ APIView để tạo thực hiện các phương thức GET, POST,PUT,DELETE

```python
class MobileApi(APIView):
```

- B9: Tạo phương thức GET

```python
@method_decorator(is_authenticated)
@method_decorator(is_admin)
def get(self, request, format=None):
    # Xử lý tìm kiếm
    query = request.GET.get('q')
    mobiles = Mobile.objects.all()
    if query == 'null':
        query = ''
    print(query)
    mobiles = mobiles.filter(Q(name__icontains=query) | Q(description__icontains=query))

    # Phân trang
    paginator = Paginator(mobiles, 10)  # Chia danh sách thành các trang, mỗi trang có tối đa 10 sách
    page = request.GET.get('page', 1)  # Trang mặc định là trang đầu tiên nếu không có trang được chỉ định
    try:
        mobiles_pages = paginator.page(page)
    except PageNotAnInteger:
        mobiles_pages = paginator.page(1)
    except EmptyPage:
        mobiles_pages = paginator.page(paginator.num_pages)

    # Tính toán totalPage và currentPage
    total_pages = paginator.num_pages
    current_page = mobiles_pages.number

    # Tạo dữ liệu cho serializer BookListSerializer
    serialized_data = {
        'mobiles': mobiles_pages,
        'totalPage': total_pages,
        'currentPage': current_page,
    }
    # Serialize dữ liệu của serializer
    result_serializer = ListMobileJson(serialized_data)
    return Response(result_serializer.data)
```

c) B10: Tạo phương thức POST

```python
@method_decorator(is_authenticated)
@method_decorator(is_admin)
def post(self, request, format=None):
    mobile = MobileJson(data = request.data)
    if mobile.is_valid():
        mobile.save()
        return Response(status=status.HTTP_200_OK)
    return Response(mobile.errors, status=status.HTTP_400_BAD_REQUEST)
```

d) B11: Tạo phương thức PUT

```python
@method_decorator(is_authenticated)
@method_decorator(is_admin)
def put(self, request, format=None):
    id = request.data.get('id', None)
    mobile = get_object_or_404(Mobile, pk=id)
    old_image_path = mobile.image.path if mobile.image else None  # Lưu đường dẫn ảnh cũ

    form = MobileJson(instance=mobile, data=request.data)
    if form.is_valid():
        # Kiểm tra xem có ảnh mới được gửi không
        if 'image' in request.FILES:
            # Xóa ảnh cũ đi
            if old_image_path and os.path.exists(old_image_path):
                os.remove(old_image_path)

        form.save()
        return Response(status=status.HTTP_200_OK)
    return Response(form.errors, status=status.HTTP_400_BAD_REQUEST)
```

e) B12: Tạo phương thức DELETE

```python
@method_decorator(is_authenticated)
@method_decorator(is_admin)
def delete(self, request, format=None):
    id = request.data.get('id', None)
    mobile = get_object_or_404(Mobile, pk = id)
    old_image_path = mobile.image.path if mobile.image else None  # Lưu đường dẫn ảnh cũ
    mobile.delete()
    #Xóa ảnh đại diện
    if old_image_path and os.path.exists(old_image_path):
        os.remove(old_image_path)
    return Response(status=status.HTTP_200_OK)
```

f) B13: Chỉ định url cho API này

```
urlpatterns = [
    path('/api-mobile', MobileApi.as_view(), name='api-mobile'),

    path('/', views.show_mobiles, name='show-mobiles'),
    path('/add', views.save_mobile, name='save-mobile'),
    path('/update/<int:id>', views.update_mobile, name='update-mobile'),
    path('/delete/<int:id>', views.delete_mobile, name='delete-mobile'),

    path('/show-types', views.show_types, name='show-types'),
    path('/add-type', views.save_type, name='save-type'),
    path('/update-type/<int:id>', views.update_type, name='update-type'),
    path('/delete-type/<int:id>', views.delete_type, name='delete-type'),

    path('/show-producers', views.show_producers, name='show-producers'),
    path('/add-producer', views.save_producer, name='save-producer'),
    path('/update-producer/<int:id>', views.update_producer, name='update-producer'),
    path('/delete-producer/<int:id>', views.delete_producer, name='delete-producer'),
]
```